New CMR ECHO Transition Plan

Original Plan Description

The original plan for transitioning ECHO capabilities to the CMR involved replacing capabilities in the kernel one by one with new capabilities in the CMR. These would be complete rewrites of the existing capabilities. We wanted to achieve the following:

- Reduce cost of maintenance
- Remove the need for the separate ECHO hardware.
- Allow easy deployment and configuration like the other applications in the CMR.
- Remove dependence on the varied technologies in those systems.
- Improve reliability and performance.
- Make it easier to improve these services.

The original plan had the major downside of requiring a lot of work and churn in our services. This would be at a time when we would be making a major transition to the cloud. We realized later that we could make the transition more easily while still achieving many of the desired goals.

Migrated Services

Access Control Service

We will complete the transition of Groups and ACLs to the CMR Access Control Service. This will provide new capabilities and an improved API for MMT, remove some of the most complex data from the Kernel, and provide other benefits as listed in the groups and ACL individual designs.

Avoiding Rewrites for Rewrites sake

We won't rewrite any other services from scratch at this time. These would be rewrites almost for rewrites sake. We would get some benefits from the CMR approach to services. However many of those benefits like easier deployments and improved availability can be achieved by taking the existing Java code and migrating mostly as is to the CMR. That's described below in SOAP Services.

Avoid Expensive Halfway Solutions

Some of these services in the Kernel would benefit from a complete overhaul and rethinking of the original use cases, for example, Ordering and Services would benefit from this. A rewrite at this time wouldn't have the resources available to make that larger kind of change possible. We might inadvertently create a new Order Service with the same limitations as the previous version. It would be better to wait until we have the resources available to devote to doing a user study, planning, and design necessary to provide the best solution.

SOAP Services

The remainder of the functionality in the ECHO Kernel will be migrated to the CMR as a standalone service called SOAP Services. The existing code would take a thin wrapper written in Clojure that allowed the Kernel code to run in our CMR environment. We have already started on this work to make it easier to test the Kernel during the transition of a rewrite to the CMR.

Benefits

- This is a much smaller amount of work than rewriting all of the existing code. We already created issues and estimated the work. There's about 2 person months of work left to complete to make the transition to SOAP Services.
- Reuses well tested code that has been continually working for most of the last 10 years.
- Provides easy deployment like other CMR apps.
- Provides better reliability by removing Torquebox as the application server which we found to be the most unreliable part of the existing setup.
- Makes it slightly easier to make improvements within the Kernel by allowing us to write Clojure integration code which can be tested in the same was as our other CMR applications.

Downsides

- Still supporting legacy SOAP APIs.
- Not as easy to improve as it would be in a new CMR service.

- We're not yet providing some of the features to providers that they may have wanted. I don't think that we would have had the resources available to do that given our current priorities. We may be able to provide some of these benefits now without a lot of work given we can make changes in the code and more easily test them after this work has been completed.
- We're leaving in some of the parts of ECHO that don't make as much sense in the current environment like security tokens and users.

REST Services

We need a transition plan for ECHO REST which is a REST API written on top of the existing SOAP API. The code for this is written in JRuby in a project called echo-rest-rails. It relies on JAXB, a Java technology, in order to efficiently create and parse XML while making SOAP requests. We don't currently have a way to deploy a JRuby application within the CMR infrastructure. We can deploy regular Ruby applications but that would necessitate making large changes to avoid the Java dependency.

Our options are the following:

- Convert to Ruby - Requires enough changes that it would be similar to a complete rewrite.
- Rewrite in Clojure - This would be a large amount of work and would take time to make it behave in an equivalent manner to the existing API.
- Make JRuby deployable via Bamboo - Not currently supported but a potentially viable option. We'd have to determine which Ruby Rack server to use with this.
- Deploy as a standalone Jar file - The Warbler tool can package JRuby applications with Rails as a standalone executable Jar or War. It will even include an embedded Jetty server.

The last two options are our easiest approaches. One of the benefits of using the last approach would be that it would be easier for us to run locally and in CI when doing integration tests. It wouldn't require starting up other tools. In any of the cases we'll have to make some changes to make it Bamboo compatible like logging via standard output and configuration via environment variables while also removing the Torquebox centric dependencies. (Not many of those since this application doesn't have database storage, caching, or background jobs)

Since we're transitioning the same code into the CMR for reuse we'll use a similar naming scheme as SOAP Services and call this REST Services. The documentation for these services will indicate its scope corresponding to the other SOAP Services.

Benefits

- We have to continue to maintain these legacy APIs. Reusing the existing code allows us to do that without a lot of work.
- We'll be able to run and deploy this API easily like other CMR applications.
- We'll get some higher reliability by moving off of Torquebox.

Downsides

- We're leaving in another technology JRuby, Rails, etc in our stack. REST Rails is using an older custom version of Rails.
- This won't address some of the performance issues we've seen with REST Rails (or the underlying SOAP Services).
    - One of the issues was in ACLs which we're completely rewriting so that won't impact us. If we have another issue with something like Ordering we'll either need to address it in REST Rails through optimizing Ruby code or rewrite just that portion in Clojure. The API has been unchanged for a long time so clients are likely working around any issues.
- It may be confusing to users how the different CMR services are structured and where they go to do different things.
    - We'll have to make sure the documentation is clear about which API to use in the new Earthdata developer portal for the CMR.
- The APIs between the older REST APIs and the newer CMR services will be inconsistent. One was implemented using Rails 5-6 years ago and the other was a different take implemented within the last 2 years.

The End Result and The Future

If we follow this plan we'll end up with the following system:

- A combination of Clojure microservices and two legacy systems, SOAP Services and REST Services.
- All of the services will be hosted in the ECC, easily deployable via bamboo, run as separate processes and configured via Hiera data.
- All of the services will run on the newer CMR virtualization hardware. The ECHO hardware can be decommissioned.

We'll address community and internal needs for things like improved data acquisition (combination ordering and services) at a later time when the need arises and resources are available.

Cloud Data Transition

We discussed alternative solutions for transitioning the data stored in SOAP Services to AWS in order to avoid having to rewrite all of the Kernel or the DAO layer of the kernel to use Metadata DB.

We looked at the counts of items in the tables and how frequently they change to understand the requirements. Most data in the kernel is very

small (< 10,000 rows) and is updated infrequently. The exception to that are security tokens (~8M and ~1 created per second) and orders (~8K orders with 1.9M order items. ~60K new order items per day)

AWS offers a new data migration service (https://aws.amazon.com/blogs/aws/aws-database-migration-service/). We tested it out and it was too slow for the initial load of granule data. The business schema is much smaller. We could use it for loading that or after initially loading the data to synchronize changes between the current operational CMR and the one running in AWS. We would probably take a very short downtime to synchronize the last of the data when we make the final switchover to redirect cmr.earthdata.nasa.gov to AWS.

We'll need to do more testing and prototyping to see if this works. Even in the case that the AWS solution doesn't work we should be able to develop a custom solution for the frequently modified data like tokens and orders to send the data up to AWS. We'll avoid having to move every DAO over to Metadata DB.